Dominig ar Foll

Senior Software Architect
Intel Open Source

IoT Summit 2016, Berlin, DE dominig.arfoll@fridu.net

Attacking IoT, a viable business

- **Ransom model**
- Stall manufacturing
- Immobilise expensive items (e.g. your car)
- …

- **Competitive advantage**
- Collecting R&D, manufacturing data
- Disturbing production line

- **Indirect**
- Cheap robot for DDoS
- Easy entry point

Understanding the risks

Developer

Fix all possible weaknesses
Deactivate possible users errors
LTS assumed for free

Back Hat

Only need one security hole
Can be help by careless users
Good long term business opportunities
Good international network

Security fundamentals

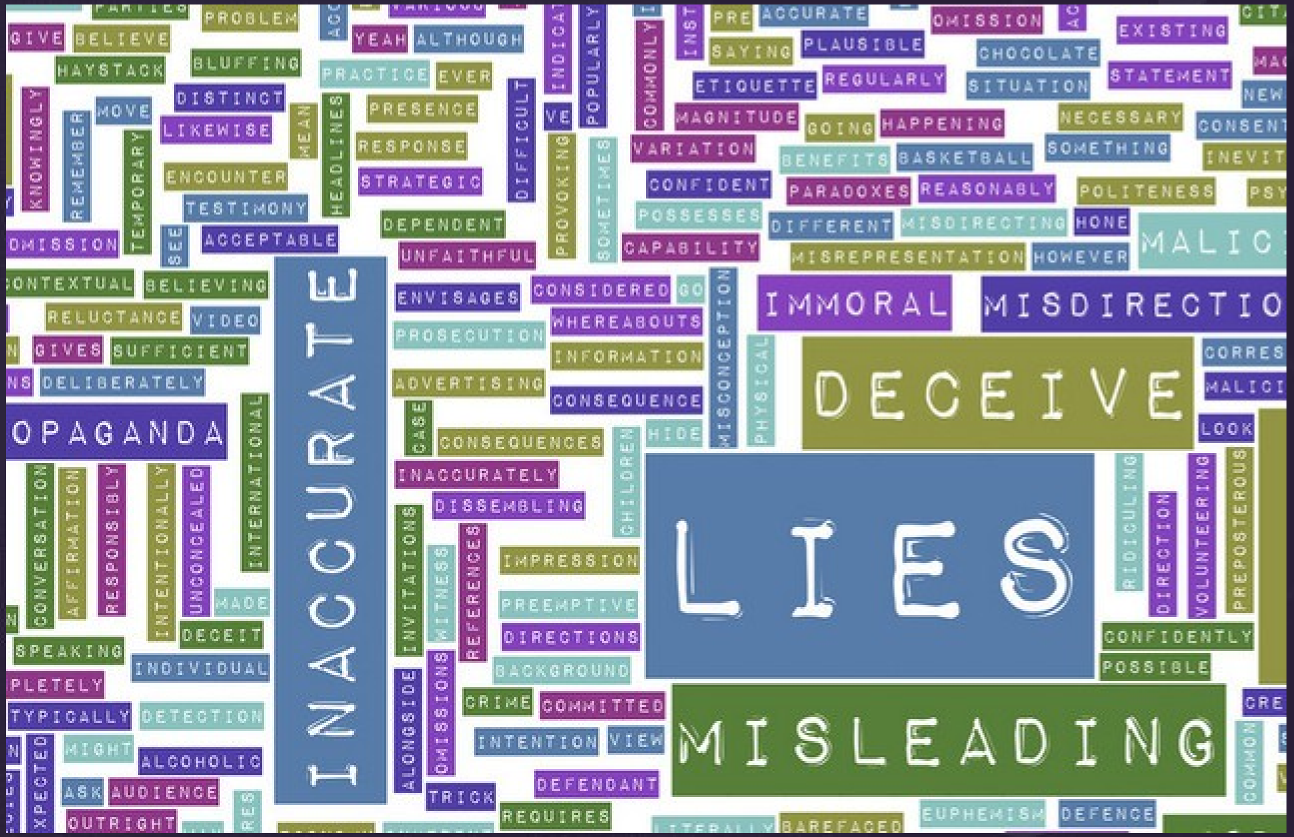
Minimise surface of attack
Control the code which is run
Provide a bullet proof update model
Track security patches
Use HW security helpers when available
Limit lateral movement in the system
Develop and QA with security turned on
Do not rely on human but on platform and tools

Security cannot be added after the fact

Do not rely on human

- **Security experts are out of reach**
- 9M Mobile Developers
- 8M Web Developers
- 0.5M Embedded Developers
- How many Embedded Security Developers ?

- **Human are unreliable**
- We do not have the time now
- Oups, it’s too late to change it
- No one is interested by our system
- We are too small
- ...



Concepts are Known but what about implementation?

AppFW
App Debug
App Packaging

Untrusted Apps / Middleware

Full isolation

API

Default policies
Debug
Sample code
HowTo

Harden OS services

Mandatory Access Control
Integrity
Name Space
Firewall
Safe update
Encryption
ID/Key protection
SoC Specific drivers

Signing
Repo create
Debug
Customize
SoC Drivers

Linux Kernel with up-to-date patches

EPID
ID Management

TPM
Private/Secure Store

UEFI
Secured Boot

Tools-Doc

Software running onTarget

Know who/what you trust

- **Trusted Boot : a MUST Have Feature**
  - Leverage hardware capabilities
  - Small series & developer key handling

- **Application Installation**
  - Verify integrity
  - Verify origin
  - Request User Consent [privacy & permissions]

- **Update**
  - Only signed updates with a trusted origin
  - Secured updates on compromised devices are a no-go option
  - Factory reset built-in from a trusted zone
  - Do not let back doors opened via containers
  - Strict control of custom drivers [in kernel mode everything is possible]

Layered Architecture

- **Client/UI (untrusted)**
  - Risk of code injection (HTML5/QML)
  - UI on external devices (Mobiles, Tablets)
  - Access to secure service APIs [REST/WS]
- **Applications & Services (semi-trusted)**
  - Unknown developers & Multi-source
  - High-grain protection by Linux DAC & MAC labels.
  - Run under control of Application Framework: need to provide a security manifest
- **Platform & System services (trusted)**
  - Message Services started by systemd
  - Service and API fine grain privilege protection
  - Part of baseline distribution and certified services only

Bullet proof update and ID

Update is the only possible correction

- Must run safely on compromised devices
- Cannot assume a know starting point

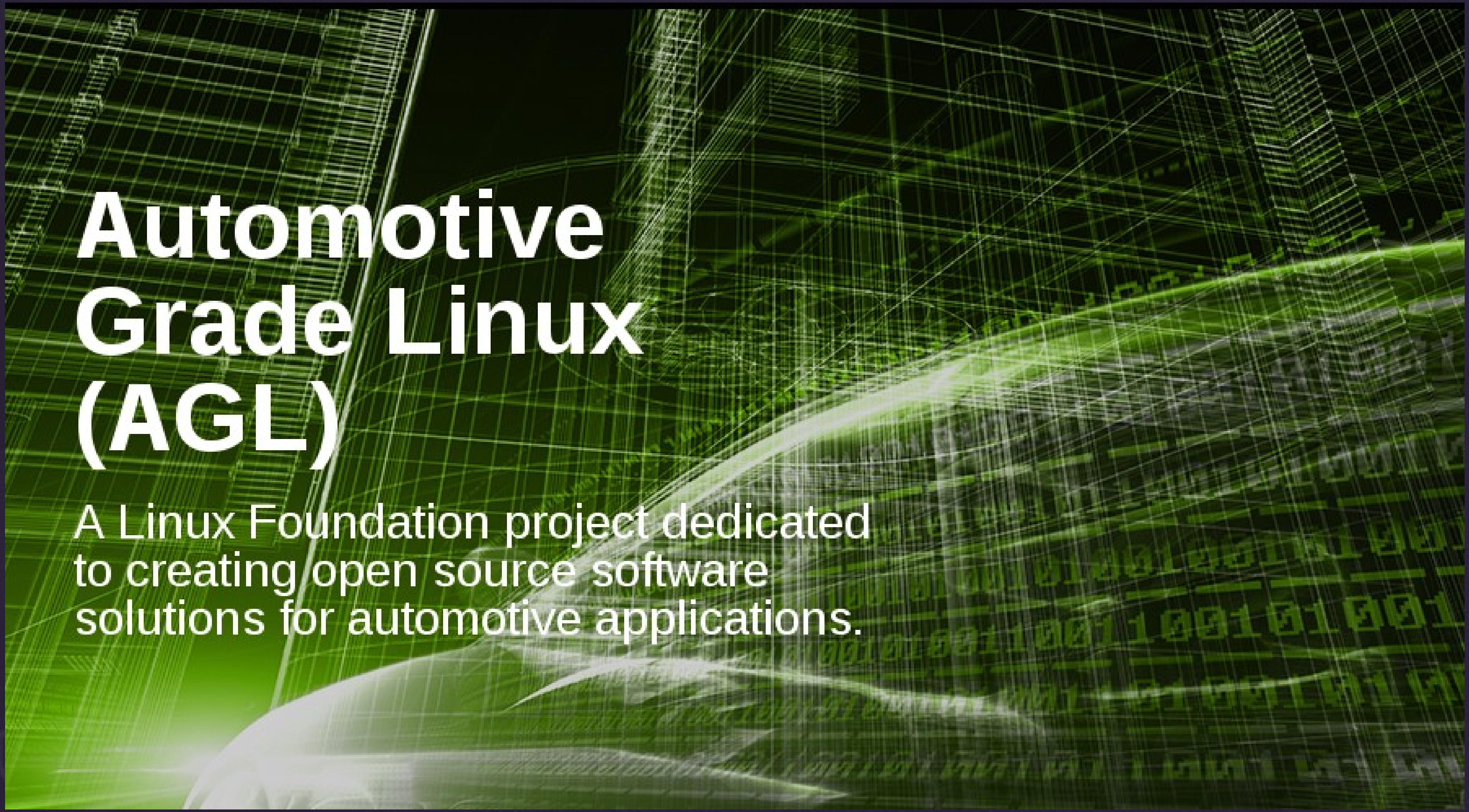
Compromised ID / keys has no return

- Per device unique ID
- Per device symmetric keys
- **Use HW ID protection** *(e.g. EPID)*

Non reproducibility

- Breaking in one device cannot be extended
- Development I/O are disabled
- Root password is unique *(or better a key)*
- Password cannot be easily recalculated

A practical example (AGL)



Applicable to any Industrial IoT Linux

Service isolation

Run services with UID<>0 SystemD is your friend

- Create dedicated UID per service
- Use Linux MAC and Smack DAC to minimise open Access

Drop privileges

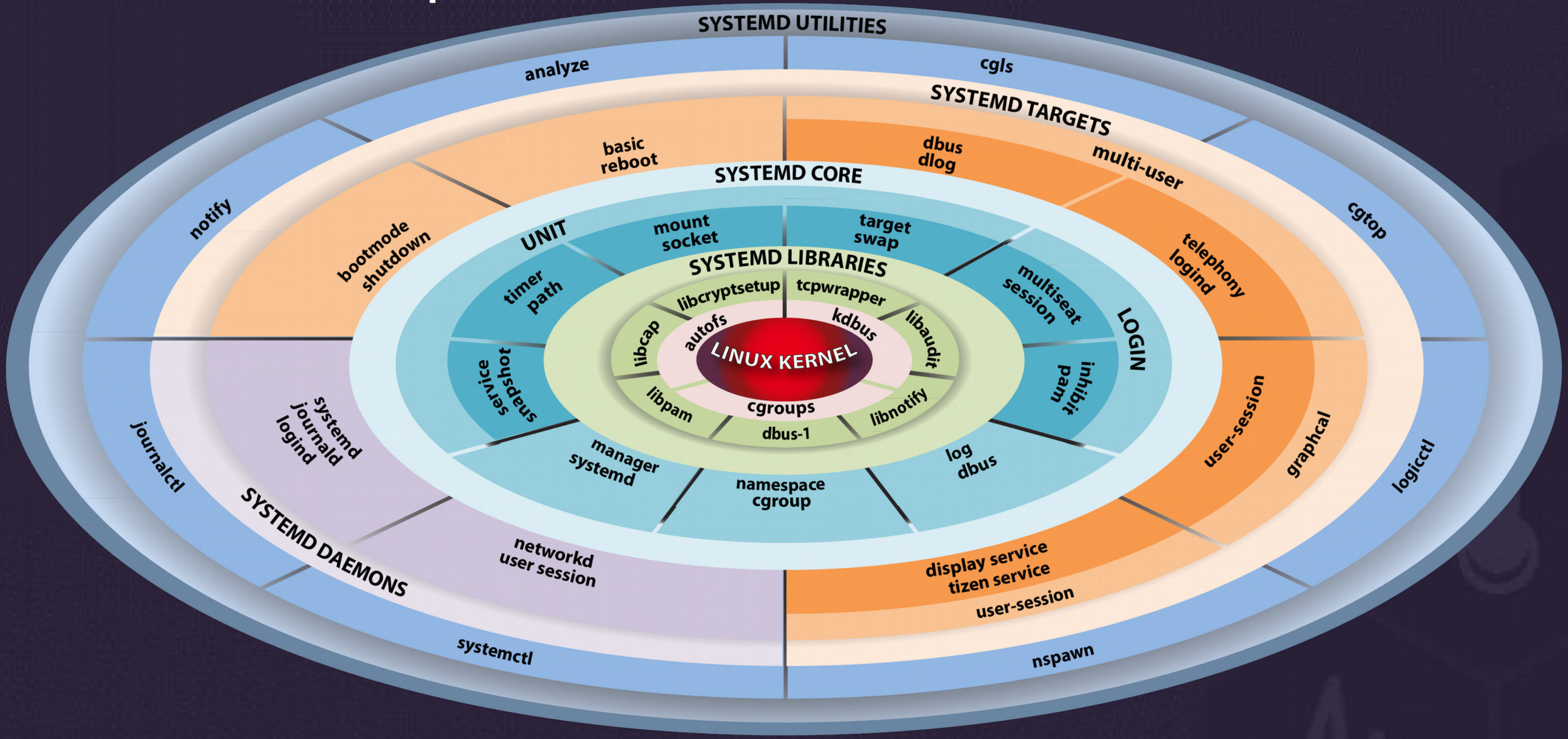
- Posix privileges
- MAC privileges

C-goups

- Reduce offending power
- RAM/CPU/IO

Name Space

- Limit access to private data
- Limit access to connectivity



https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt
https://www.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.2/capfaq-0.2.txt
http://man7.org/linux/man-pages/man7/namespaces.7.html
https://en.wikipedia.org/wiki/Mandatory_access_control
https://en.wikipedia.org/wiki/Discretionary_access_control

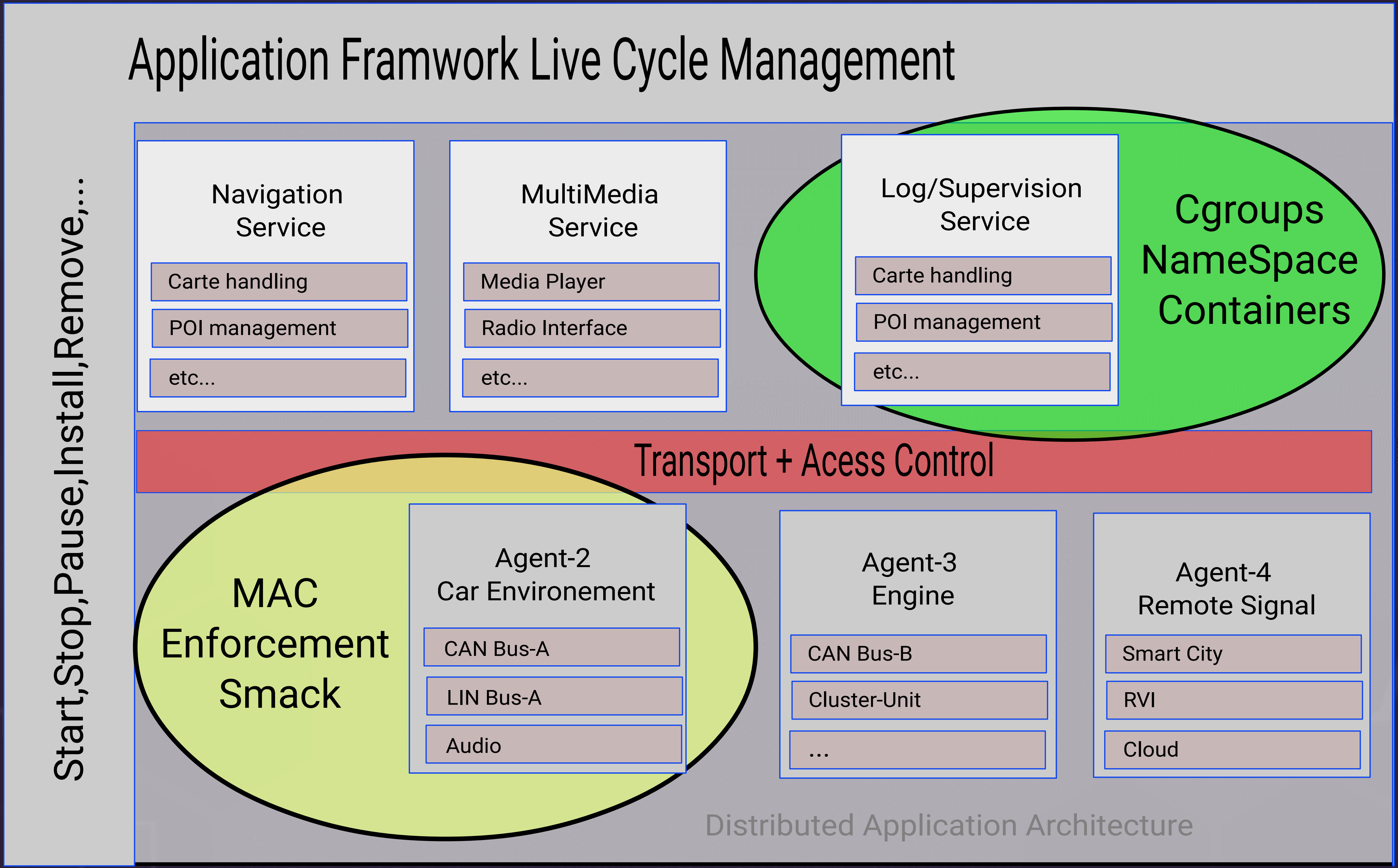
Segregate Apps from OS

- **Application Manager**
  - One system daemon for application live cycle installs, update, delete
  - One user daemon per user for application start, stop, pause, resume
  - Create initial share secret between UI and Binder
  - Spawn and controls application processes: binder, UI, …

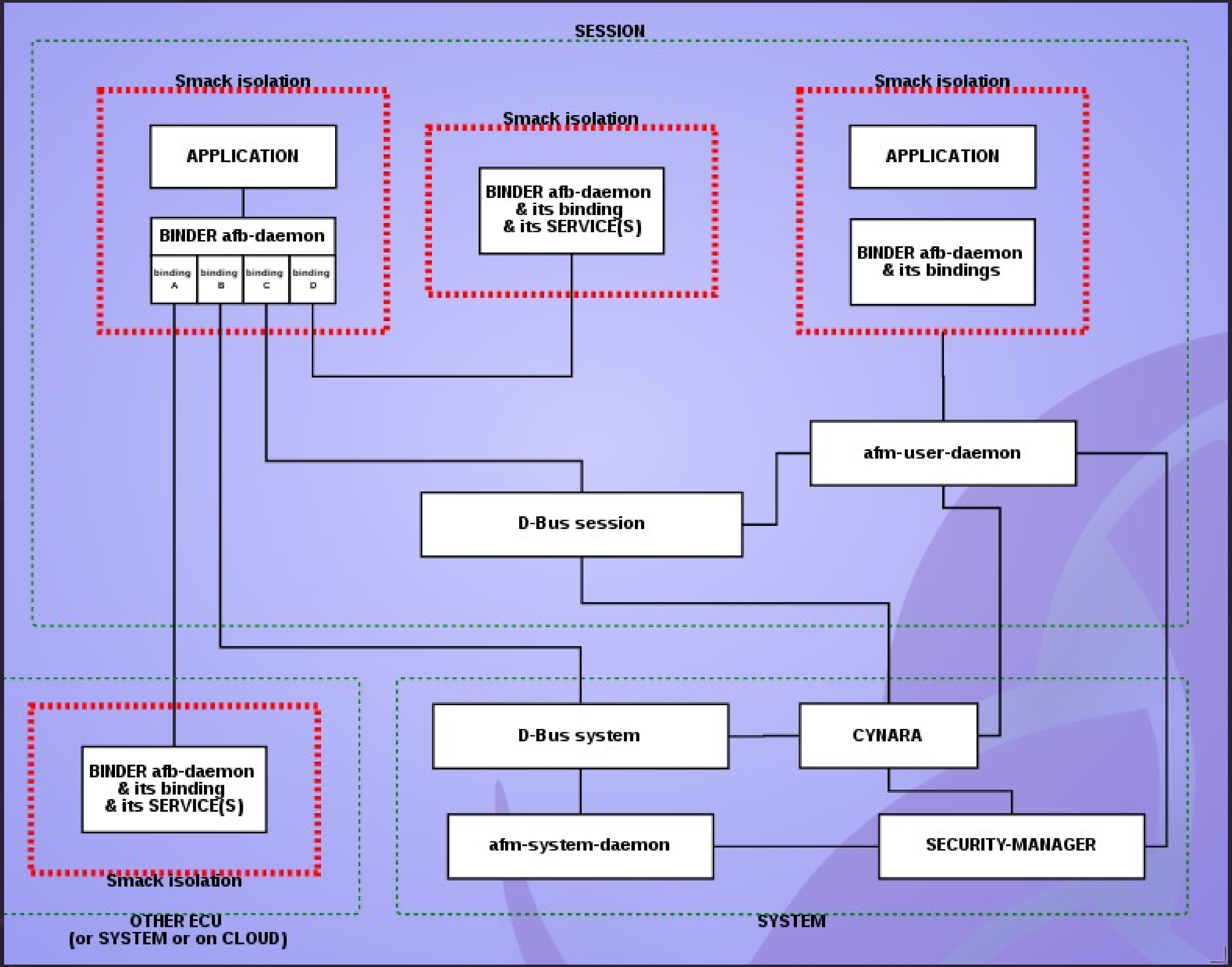
- Security Manager
  - Responsible of privilege enforcement
  - Based on Cynara + WebSocket and D-Bus for Legacy)

- **Application & Services Binders**
  - Expose platform APIs to UI, Services, Applications
  - Loads services/application plugins :Audio, Canbus, Media Server…
  - One private binder per application/services [REST, WebSocket, Dbus]
  - Authenticate UI by oAuth token type
  - Secured by SMACK label + UID/GIDs
  - AppBinders runs under user $HOME

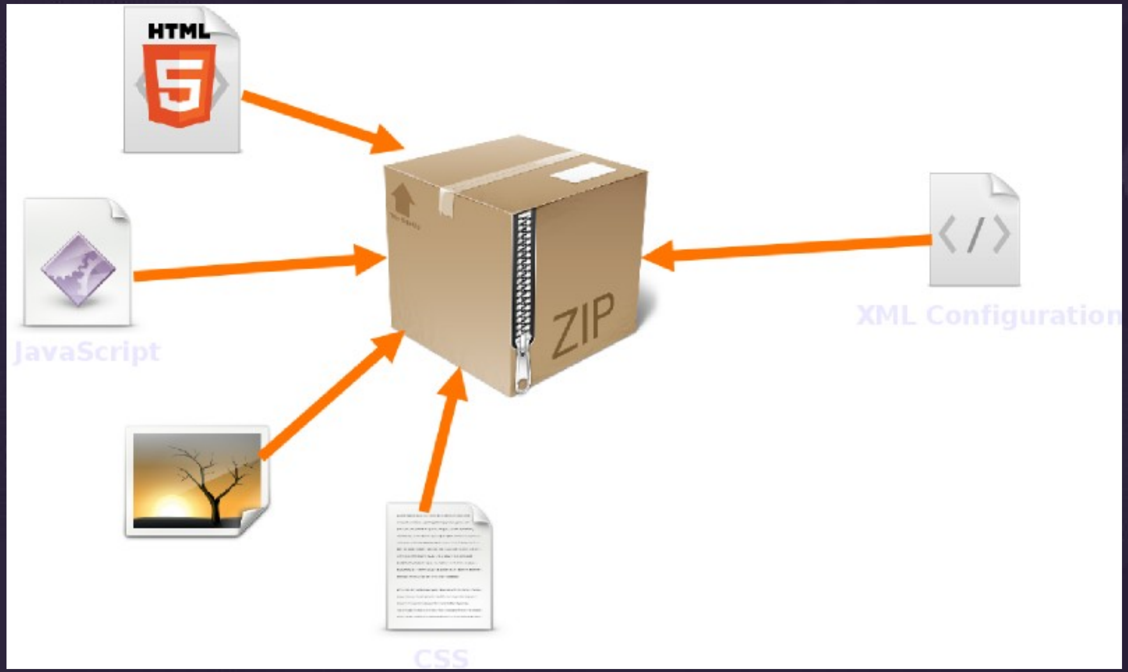
AGL2 Application Security



AGL2 AppFW logic

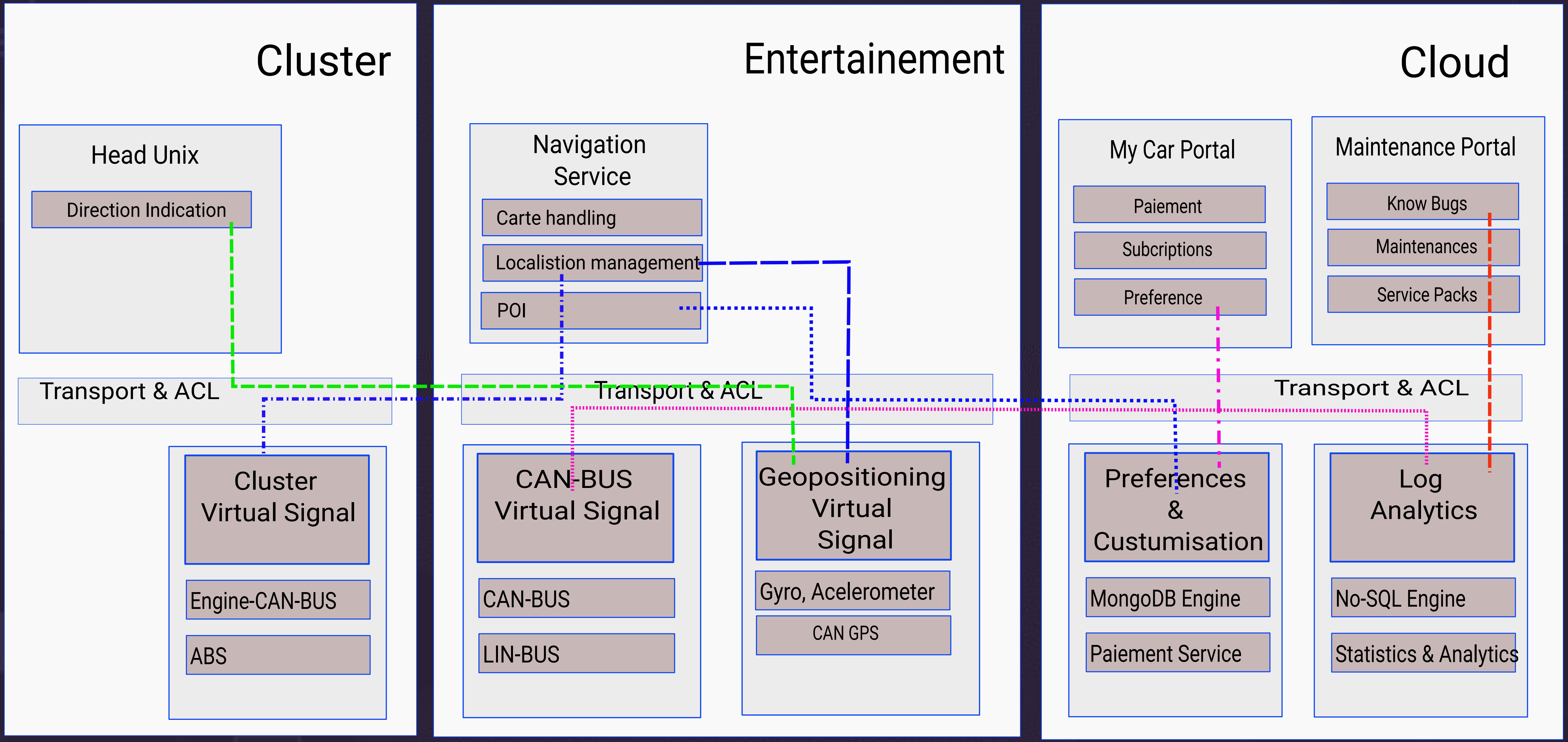


To write an App



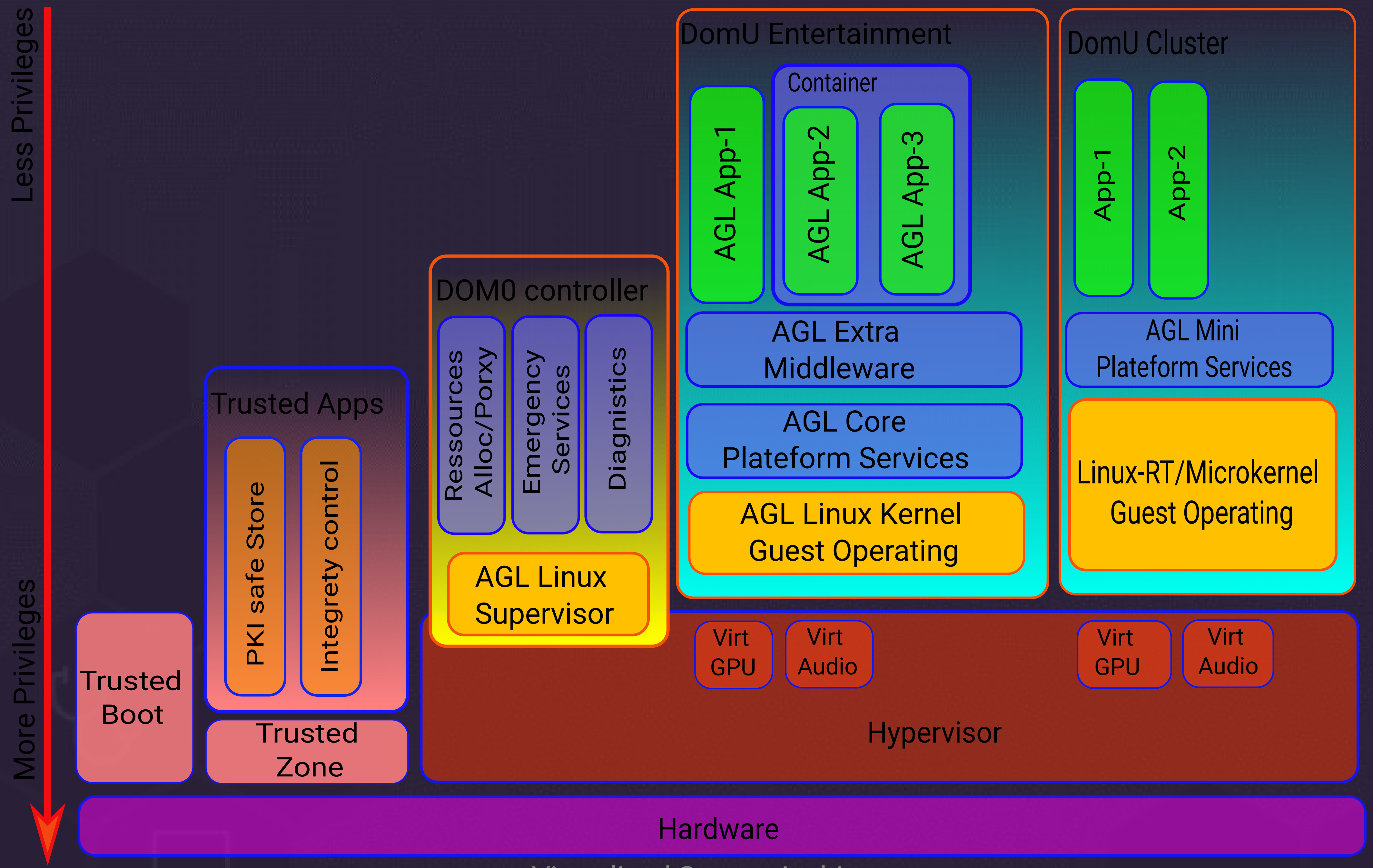
- **Write back-end binding**
  - Adds the specialised API to the system
  - Accessible by Web Socket or slow legacy D-Bus
  - Run in its own security domain
  - Can be cascaded
- Write the Front end
  - Typically in HTML5, QML but open to any
  - Connect to back-end binding using REST with secured key (OAuth2)
  -
- **Package**
  - Based on W3C widget
  - Feature allow to handle AGL specificities
  - Install via the AppFW

AGL2+ Distributed Architecture



Multi ECU & Cloud Aware Architecture

AGL2++ Virtualised Architecture

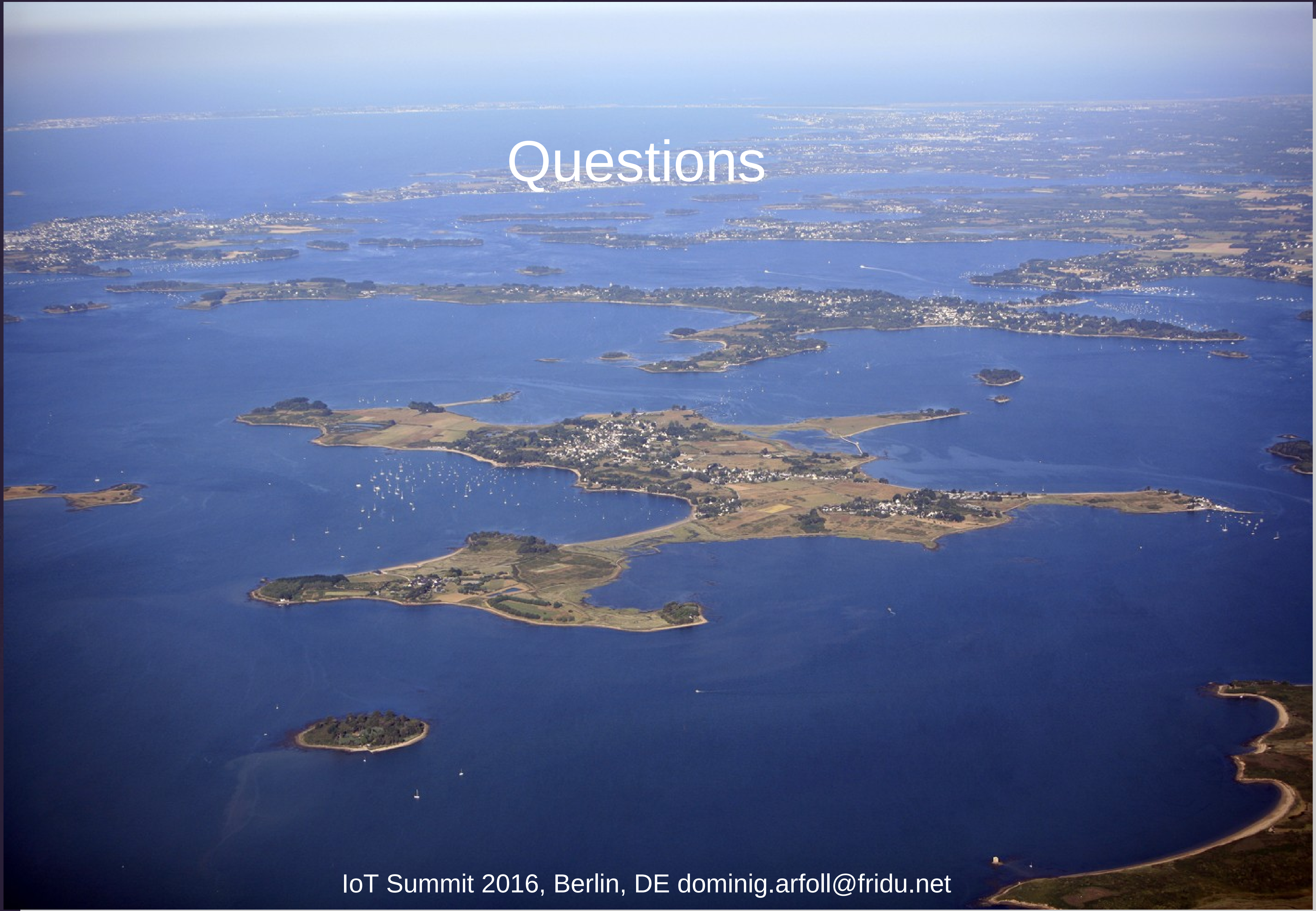


Virtualized Secure Architecture

Conclusion

- **Technologies are available**
  - Secure boot, Secure zone
  - Update over the air
  - Isolation and containment
  - Tools and training
- Management is not ready
  - Still perceived as a nice to have
  - Too risky to commit
- **Engineering sees security as a brake to innovation**
  - Requires a serious personal investment and paradigm shift
  - Complexity imposes to select a “Ready Made” solution
    - AGL, Tizen, Snappy, ...
  - “Will add it later” attitude is common but a guaranteed model to failure

Questions



Container "A mixed blessing"

Easy to use

- Detach the App from the platform
- Integrated App management
- Well known

Not very secure

- Unreliable introspection
- MAC has no power on the inside of a container
- Updating the platform does not update the middleware
- Beside the Kernel each App provide its own version of the OS
- Each App restart requires a full passing of credential
- RAM and Flash footprint are uncontrollable
- Far more secured with Clear Container but not applicable to low end SoC.

Only I/O via network

- Well equipped for Rest API
- All other I/O requires driver level access or bespoke framework.

https://www.opencontainers.org/
https://lwn.net/Articles/644675/

Security Check list

Control which code you run
- Secure boot
- Integrity
- Secure update

Isolate services
- Drop root when possible
- Drop privileges

Isolate Apps
- Apps are not the OS
- Enforce – restrict access to standard API

Identity
- Enforce identity unicity
- Use available HW protection

Encryption
- Network traffic
- Local storage

Control image creation
- No debug tool in production
- No default root password
- No unrequired open port

Continuous integration
- Automate static analysis
- QA on secured image

Help developer
- Integrate security in Devel image
- Provide clear guide line
- Isolate Apps from OS
- Focus on standardised Middleware